SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 6, 271–279 (1976)

The Design and Implementation of an Interactive Document Editor

G. F. COULOURIS, I. DURHAM, J. R. HUTCHINSON, M. H. PATEL, T. REEVES AND D. G. WINDERBANK
Computer Systems Laboratory, Department of Computer Science and Statistics, Queen Mary College, University of London, England

SUMMARY

General-purpose text editors have significant drawbacks when used for document preparation. This paper describes a system designed to combine the editing and formatting power of a text editor with the simplicity and immediacy of a typewriter for document preparation. The system is implemented on a minicomputer with a simple alphanumeric visual display unit.

KEY WORDS Editor Document preparation Word processing Reactive system Command language

INTRODUCTION

This paper describes the design and implementation of QUIDS (a QUick Interactive Documentation System), a text editing program developed in response to the following question: how can the information processing capacity of a small computer best be exploited to simplify the tasks of typing, formatting and printing documents such as manuals, reports, papers and theses?

Many systems have been developed and used for this purpose in time-sharing systems. Reference 1 contains a survey of some of them. Their use to date, however, has been largely restricted to those computer professionals and others who are able to apply that combination of know-how, access privilege and doggedness required to achieve the desired result.

Our experience with a conventional text editor (very similar to QED[2]) running on a dedicated small computer with an interactive operating system[3] had indicated that the attributes of computer professionals listed above are not necessarily essential to the successful use of text editing systems for document production.

But QED and most other text editors are designed to operate with text files consisting of a sequence of lines of text. The archetypal example of such a file is the source language computer program.

Although printed documents appear to conform to a similar structure, this similarity is misleading, since the position of line breaks in document texts is almost entirely arbitrary, and the user does not normally wish to be concerned with them. The use of a line-oriented editor to create and modify the document text is clearly incompatible with this desire, and leads to considerable difficulty when more than minor alterations are made to the text.

This problem is normally overcome in time-sharing systems by the use of a separate program that will accept a file containing lines of text with additional formatting information, and print the text in the required format without regard to the position of line breaks in the source file. A widely used example of this type of program is the Runoff system.[4]

Received 12 May 1975
Revised 21 July 1975

Unfortunately two further problems arise for the user of this type of system. First, he still prepares his document text using a conventional editor as a sequence of lines, some containing formatting information and the remainder containing the text. However, the lines in the file he prepares no longer bear any particular relationship to those in the printed document. This makes revisions and corrections inconvenient. Second, before he can use a Runoff-type system, he must be conversant with a fairly extensive formatting notation. Furthermore, errors made in describing the layout of a document when preparing it will not become apparent until the full text has been prepared and submitted for Runoff processing. It may take several cycles of editing and processing before a satisfactory result is produced in this way.

QUIDS overcomes the first of the above problems by designating the paragraph as the basic unit of stored text. This enables the user to generate and modify his text without regard to line breaks.

The second problem is minimized in two ways. First, the format description for a document is generated as a part of an interactive dialogue between the user and QUIDS. Many types of error can be detected and indicated by QUIDS at this stage. Second, printing and display of the formatted text is a function of the editor. This enables the user immediately to view the effects of insertions, alterations and layout changes, in either a local context or a broader one.

FUNCTIONAL REQUIREMENTS

In many respects, the typewriter is a satisfactory instrument for the preparation of printed text. Its deficiencies are apparent where corrections, insertions and revisions are concerned and in the minimal assistance that it affords the user in laying out his text.

In designing a system intended to remove the deficiencies of the typewriter, we were concerned to retain all of those simple yet essential features that have made the use of the typewriter so universal. These include the facility to view the text continuously and with whatever range of context is desired. Alterations, although clumsy on a typewriter, produce an updated page immediately. Continuous availability of the machine for keyboard input is also something that cannot be taken for granted in the computer environment.

The above requirements, together with those discussed in the preceding section, imply performance criteria for a document editor and its supporting hardware. The most critical of these are the ability to locate an arbitrary string within an entire document and to display a page of text in its formatted form, without inconvenient delays. It is also important to maintain a correct and up-to-date display of text as local alterations are made.

HARDWARE

At the cost of a fairly complex keyboard input program, it has been possible to provide comprehensive facilities for alterations to text while maintaining a correct display of the relevant text throughout. This is done using a teletype-compatible visual display interfaced to the processor at a data rate of 9,600 bits per second (i.e. nearly 1,000 characters per second).

A conventional minicomputer (Interdata 70) having 48K bytes of main memory is used to perform the editing and formatting operations directly on text held in main memory. A cartridge disk system is also attached and a filestore is supported by the operating system. A fast character printer (100 characters per second) with a 96-character fount is used to

produce formatted hard copies. Higher quality copies are produced with an IBM Selectric typewriter terminal or a Diablo Hityper. The hard copy devices are shared between several computers.

This choice of hardware was made in the knowledge that although on current costs the resulting system might be difficult to fully justify, trends in electronic hardware costs and increased use of networking as a means of sharing mechanical peripherals are likely to make similar systems entirely justifiable in the near future. QUIDS can be seen as a software prototype for such systems.

TEXTUAL CONSTRUCTS

A document text is composed as a sequence of paragraphs and similar items of text (blocks, tables, displayed strings), and associated titles, main section headings and subsection headings. In addition format control items may be associated with text items, to produce variations in layout within the text (margin indentation, vertical spacing, figure inserts, labels in the left margin, etc.).

QUIDS enables the user to create, delete, replace, modify, display or print individual items or sequences of items in a document text. These actions are performed using interactive commands in the manner discussed in the next section. The principal commands available in QUIDS are listed in Appendix 1.

A further level of control is associated with the generation of new text items. At this level, each type of text and format item is identified by an initial letter (see Appendix 2). Within each such item, the normal keyboard is used to enter the actual text. During text entry, and subsequently for corrections a comprehensive set of editing facilities is supported, using special function keys, or where these are not available, control-key combinations (Appendix 3).

INTERACTION

Each character typed at the keyboard is either a contribution to the text of a document in preparation, or else it specifies a function to be performed by the editor. In the latter case, the range of commands and the notion of 'mode of interaction' are specified so that a single key depression provides sufficient information for QUIDS to interpret the function required. The character typed is interpreted as the initial letter of one of a standard set of commands. The remainder of the command name is concatenated to the initial character and displayed by the editor as a confirmation to the user. In most cases (except where the action may be irreversible) the function specified is then performed without further input from the user.

Errors in the formation of commands to QUIDS are also detected and indicated visually and aurally (by a bleep) to the user as soon as they occur. Full error descriptions are displayed in response to a '?'.

While the user is entering his text and correcting it, the aim of the system is to maintain an up-to-date display of the relevant text as he deletes and alters words, characters or strings.

The choice of a conventional teletype-compatible VDU might be seen as in conflict with this aim. However, it is possible to overwrite the last line displayed on such terminals, replacing the old line with the corrected one. When this is done at a speed approaching 1,000 characters a second, it gives the illusion of an instantaneous correction.

The mode of interaction described here operates satisfactorily for users of all degrees of expertise provided that the time taken to display command names and corrections to text is comparable to the normal delay between key depressions. The full record of commands and text produced on the display is of course useful to the user during a sequence of interactions.

AN EXAMPLE

It is possible here to illustrate only what is produced on the display screen during part of an interactive session. The user is prompted to enter commands by a '>' or item names by a single quote. When he types a correct initial letter the remainder of the prompted line is generated by the system. Numbers (e.g. '1.0.2') are generated by QUIDS to assist the user in subsequent reference to the items. Text items are of course entered in full by the user, terminating with an 'escape' which is not visible. Line breaks are generated on screen automatically at appropriate word boundaries during text entry, but are not the retained in the stored text.

Example

```
>Append
'Call it :
Interactive Document Editor (ESCAPE)
'Begin new page
'Main heading 1:
Introduction (ESCAPE)
'Paragraph  1.0.1:
This paper describes the design and implementation of QUIDS (a _QU_ick
_I_nteractive _D_ocumentation _S_ystem), a text
. . . . . .
for document production. (ESCAPE)
'Paragraph  1.0.4:
But QED and most other text editors . . . . . .
'Paragraph 1.0.5:
Although printed documents appear to conform to a similar structure,
    ⋮
>Options
*Title each page (no) [y or n]: Y
*Numbered sections (yes) [y or n]: N
*(ESCAPE)
>,/
```

INTRODUCTION

This paper describes the design and implementation of QUIDS (a QUick Interactive Documentation System), a text ....
Many systems have been developed and used for this purpose ....
Our experience with a conventional text editor (very
⋮
(end of page)

INTERACTIVE DOCUMENT EDITOR PAGE 2

similar to QED[2]) running on a dedicated small computer with an interactive operating system[3] had indicated ....
But QED and most other text editors are designed to
⋮

CONTEXT AND SCOPE

As in all text editing systems, it is important to include convenient facilities for the user to refer to items that require modification. Two methods are available in QUIDS. First, the user can perform a search for an arbitrary string of characters to locate the next occurrence of the string in the document text. Alteration of the string once found can be performed either as a part of the same action (using the *xchange* command), or as a separate sequence of interactions.

To delimit the scope of such searches (and other commands that may be applied to more than one item), item numbers are used. Commands are prefixed by one or two item numbers. If either is absent then default values are assumed (normally the numbers of the first and last items in the text). Thus the commands may have unlimited scope, yet the user can exercise complete control where necessary.

The numbering system used is intended to approximate to that used in manual document preparation. Thus the three integers in a number of the form 1.0.2 give respectively the section, sub-section and paragraph number for the item referenced.

As an example of the use of context delimiters, consider the following command:

1.0.2, 2.1.3 delete [*confirm*].

On receipt of the confirmation character (i.e. '.', necessary only for irreversible commands such as *delete*), the editor removes those items having numbers between *1.0.2* and *2.1.3* from the current text. Since numbering is computed by QUIDS from the actual text on a dynamic basis, item *2.1.4* becomes item *1.0.2* after the above action.

If the section number of an item to be altered is known to the user, then he can use it to refer to the item without a search. This is the second of the two techniques referred to above.

However it is frequently unnecessary to quote section numbers. The notion of a 'current item', together with the use of default values enables most editing to be performed without them. In the final text, section numbers are printed only if required.

IMPLEMENTATION

The implementation of QUIDS was undertaken by the authors as a student team project based on a functional specification prepared by one of us.

A simple stored data structure was formulated at an early stage. This has proved adequately efficient, and its simplicity is an aid to the specification of independent program modules to perform the editing and other functions of QUIDS.

The text of a document is read from a file and held in main memory throughout the editing process. (Simple facilities for the segmentation of long documents are included. No segment may exceed 16,000 characters in length.) The specification implies three basic classes of item:

paragraphs—	the basic units of text manipulated by the editor;
format specifications—	these affect the layout of one or more following paragraphs;
non-sequential text—	some text items are not a part of the main sequence (and do not receive item numbers), e.g. page titles, footnotes and reference entries.

Each type of item is represented by a simple stored data structure containing a record length, an item type code, a 'countability code' and text or parameters depending on item type. No further data is stored other than working variables and temporary buffers for local editing operations. The three-part numbers associated with items and all of the format information (page numbers, current margin position, etc.) are computed as required.

The text items are stored as a strictly contiguous sequence of structures as described above. The insertion of new items and deletion of existing items therefore results in the movement of all following items in the memory. The time taken to do this is not noticeable to the user at the terminal.

The 'countability code' is used to assist in computing item numbers. The record length makes sequential processing of the text relatively simple.

The display and printing of formatted text with right margin justification, indentation and all format controls required by the user is performed 'on the fly' from the stored representation. Since the format of any part of the text is in general dependent on its context (for the degree of indentation, position of page breaks, etc.), a command to display any part of the text demands a scan of the whole.

The program structure of the editor falls naturally into the following modules:

an interactive command handler;
a keyboard input and local editing procedure;
a text formatting and printing procedure;
a file transfer procedure;
a context searching procedure.

The first three of these account for the bulk of the code. Each module constitutes a disk overlay, with overall control residing in the command handler.

The program is written in a high-level system implementation language with machine language segments where necessary for reasons of speed or language restriction. It occupies about 20K bytes of main memory excluding the text data structure. Including overlays, the total volume of code is about 60K bytes.

CONCLUSIONS

Despite the rather advanced specification, QUIDS has been successfully implemented with the required performance in the chosen hardware configuration. This has been achieved without the use of any especially complex or obscure programming techniques. We believe that the availability of dedicated processing power has in many cases removed the need for such techniques. The editor executes most commands within 1 second. There is no delay in updating corrected text on the display. Context searches and formatting operations can take up to 10 seconds on long texts, but the overall effect achieved by the editor is to give the user the impression that actions are performed on demand, as if by the hardware.

Many documents have now been prepared with the aid of QUIDS. Users, including secretarial staff, students and academic staff, many of whom have access to alternative facilities of the Runoff type, are able to produce a polished result more quickly using QUIDS. Some even claim that their powers of expression have been enhanced by the increased flexibility in manipulating the text.

On the other hand, users have suggested, and we have observed ways in which the specification could be bettered. The idea of 'modes of interaction' is not as natural to the user as we should like. Of course, this notion enables the range of commands to be far

greater than the set of available initial letters. No other method for suitably structuring the fairly extensive command language has yet suggested itself to us.

Local text editing causes little difficulty once the range of control functions has been learned, but the use of a more powerful display allowing the user to point to strings would undoubtedly reduce this learning time and make the revison of text more like the use of the familiar 'blue pencil'. It might also enable the command language to be sufficiently simplified to reduce the number of modes of interaction to one.

This observation, and the lack of suitable terminals in the market-place, has led to a recent study at Queen Mary College into the design and construction of a terminal with comprehensive facilities for text manipulation and display.

The cost justification for the entire system is of interest. At the time of writing, the use of a dedicated minicomputer system appears to be only slightly more costly than the apportioned cost of the resources required to support a similar application at a lower level of performance in a time-sharing system. We shall not attempt to extrapolate from the current position, but we believe that in some circumstances, this extra cost can be justified by the additional performance of a dedicated system.

QUIDS was developed by a team of five undergraduate students and one staff member in four or five months. We estimate that the scale of effort involved is two to three times that required to implement a conventional line-oriented text editor. (But only slightly greater than that for an editor plus a Runoff program.)

ACKNOWLEDGEMENTS

The project was carried out using the facilities of the Computer Systems Laboratory at Queen Mary College. We are grateful to the College, its Department of Computer Science and Statistics, and in particular to all of those past and present members of the Computer Systems Laboratory who had earlier developed and made available basic software tools that were essential to the completion of the project. We should also like to thank those members of the Laboratory who contributed their suggestions and assistance during the design and implementation of QUIDS.

We should like to thank the referee for his perceptive and constructive comments on the first draft of this paper. With the assistance of QUIDS, we were able to incorporate his suggestions without difficulty.

APPENDIX 1: CONTEXT EDITING COMMANDS

Any of these commands may be used in response to the '⟩' prompt. In the following *N*, *M* are any one of section, subsection or paragraph numbers (i.e. *a* or *a.b* or *a.b.c* respectively). Default is usually the current item. Keyed characters are shown in italics. The case is not significant.

*N A*ppend	Enter ' mode after item *N*.
*N I*nsert	As above, but before item *N*.
*N, M D*elete [confirm]	Delete items *N* through *M* after '.' confirmation.
*N, M C*hange [confirm]	As above, but enter ' mode to replace deleted items.
*N, M E*dit text	Enter local edit mode for each paragraph in items *N* through *M*.
*N, M F*ormat change	Insert, Append and Delete editing of format items in the given range.

N, M !	Unformatted display of items *N* through *M* on the screen.
N, M	As above, but formatted.
N, M Print on ⟨*file device*⟩	Formatted output to file or device.
N Read from ⟨*file*⟩	Reads text from file after item *N*.
N, M Write on ⟨*file*⟩	Write items *N* through *M* to file.
N, M Xchange /*s1*/ for /*s2*/	Replaces each confirmed occurrence of string *s1* by *s2* in the given range. '.' is confirmation character.
Options	Enter * mode to select options.
Table of *R, C* or *I*, on ⟨*file*⟩	Produces table of References, Contents or Index items on file.
Quit (and save text [y or n] on ⟨*file*⟩	Leaves QUIDS after saving text.
;*s1*;	Moves to next item containing *s1*.
N '*linefeed*'	Unformatted display of the paragraph after item *N*.
N ^	As above, but before item *N*.
? or *CTRL + H*	Help with mistake or command list if no mistake made.

APPENDIX 2: TEXT INPUT COMMANDS

These commands are used in response to the ' prompt. In the following, QUIDS prints the item number as applicable.

*M*ain heading	Main section heading. (Given next section number.)
*S*ub-heading	Sub-section heading. (Given two-part number.)
*P*aragraph	Enter normal paragraph.
*E*ntry	Paragraph without first line indent.
*N*on-formatted	Paragraph not to be processed by QUIDS formatting.
*D*isplayed text	Block of text centred on the page (e.g. a book extract).
*L*eft	Item to be printed in the margin (space allowing).
*C*all it	Document and/or page title.
*R*eference =⟨*n*⟩. [. . .]	Bibliographic reference number ⟨*n*⟩. [. . .] string replaces [*ref. n*] in document.
*B*egin new page	Skip to new page.
*V*erical =⟨*n*⟩	Skip ⟨*n*⟩ lines.
*W*idth =⟨*n*⟩	Set right margin ⟨*n*⟩ characters from left margin.
*I*ndent =⟨*n*⟩	Indent left margin ⟨*n*⟩ spaces.
*U*ndent	Repeal last indent.
*F*igure =⟨*n*⟩ size is =⟨*m*⟩ chars by =⟨*l*⟩ lines (left or right)	Reserve numbered figure space. Appears after [*fig. n*] in the document.
Tabset =⟨*n1*⟩, =⟨*n2*⟩, ... =⟨*ni*⟩	Set tab stops for non-formatted paragraphs.

APPENDIX 3: INPUT/LOCAL EDIT FUNCTIONS

('^' indicates control-shift characters)

Exit (^*D*)	Complete the edit/insert, & return to command level.
Cancel (^*X*)	Cancel the current insert/edit, and restore the previous version if any.

Forward word (^*A*)	Move forward one word.
Move to end (^*E*)	Move to the end of the current item.
Delete marked text (^*F*)	Delete all text between the two delete flags previously set.
Help (^*H*)	Give help with errors.
Set delete flag (^*N*)	Set a flag at the current position (see Delete above)
Move to line start (^*O*)	Moves back to the start of the currently displayed line.
Print (^*P*)	Displays from the start of the current item to the current position, clearing the screen.
Move one character (^*Q*)	Move forward one character.
Move to start (^*S*)	Moves to the start of the current item.
Back one word (^*T*)	Move back one word, and redisplay line.
Delete word (^*Z*)	Deletes the last word displayed.
Rubout	Delete the last character displayed.
Linefeed	Move forward one line, and display it.

REFERENCES

1. A. Van Dam and D. Rice, 'On-line text editing: a survey', *Computing Surveys*, **3**, 93–114 (1971).
2. P. Deutsch and B. W. Lampson, 'An on-line editor', *Comm. ACM*, **10**, No. 12, 793–799 (1967).
3. M. S. Cole and J. Rowson, 'An environment for software development using minicomputers', available as Computer Systems Report No. 2, Computer Systems Laboratory, Queen Mary College, 1975.
4. Bolt, Beranek and Newman Inc., *Tenex User's Guide*, 1973, p. 141.